

FIG. 1

	100	110	120
	Acme 1.0 Persistent Store	Acme 2.0 Memory	Acme 2.0 Persistent Store
150	Class Widget1 (1.0)		Class Widget1 (1.0)
160	Class Widget2 (1.0)	Widget2 → Widget2-Redesigned 180	Class Widget2-Redesigned (2.0)
170	Class Widget3 (1.0)		Class Widget3 (1.0)

2/27

FIG. 2

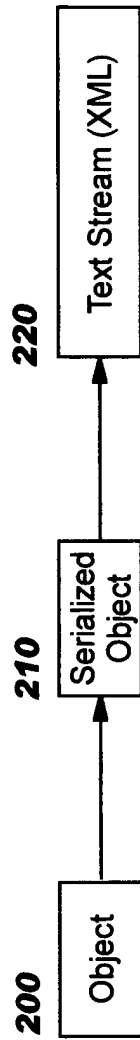


FIG. 3  
(Prior Art)



3/27

FIG. 4

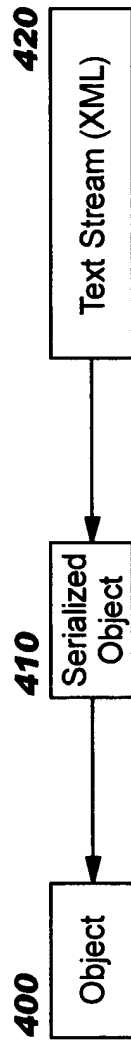


FIG. 5  
(Prior Art)



4/27

FIG. 6

```
<?xml version="1.0"?>  
  
<company name="Acme" url="http://acme.com">  
  <employee name="Paul" officeNumber="400"/>  
  <employee name="John" officeNumber="500"/>  
</company>
```

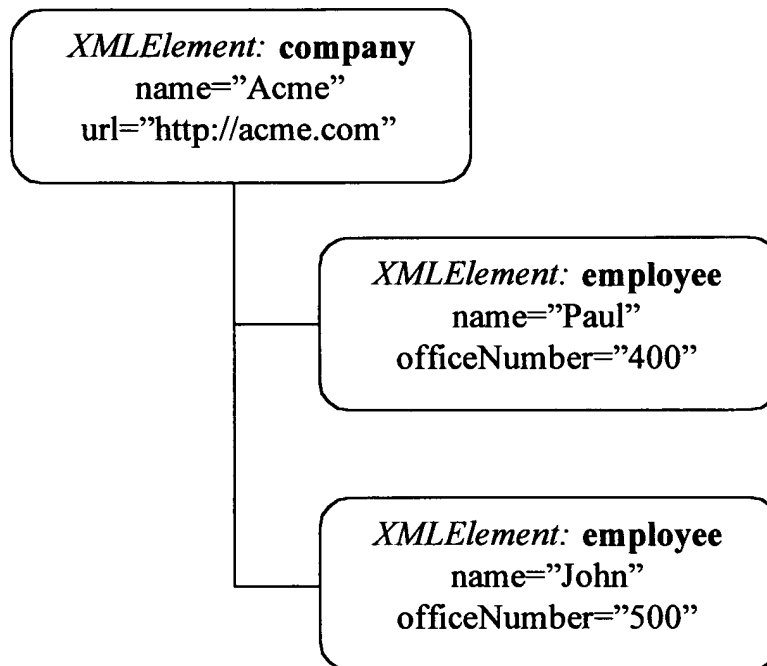
FIG. 12

```
<?xml version="1.0"?>  
  
<object name="bill" className="Employee">  
  
  <class name="Person" serialVersion="12345">  
    <special-object name="name" className="java.lang.String" value="William"/>  
    <primitive name="age" className="int" value="20"/>  
  </class>  
  
  <class name="Employee" serialVersion="67890">  
    <special-object name="title" className="java.lang.String" value="Engineer"/>  
    <primitive name="officeNumber" className="long" value="117"/>  
  </class>  
  
</object>
```

5/27

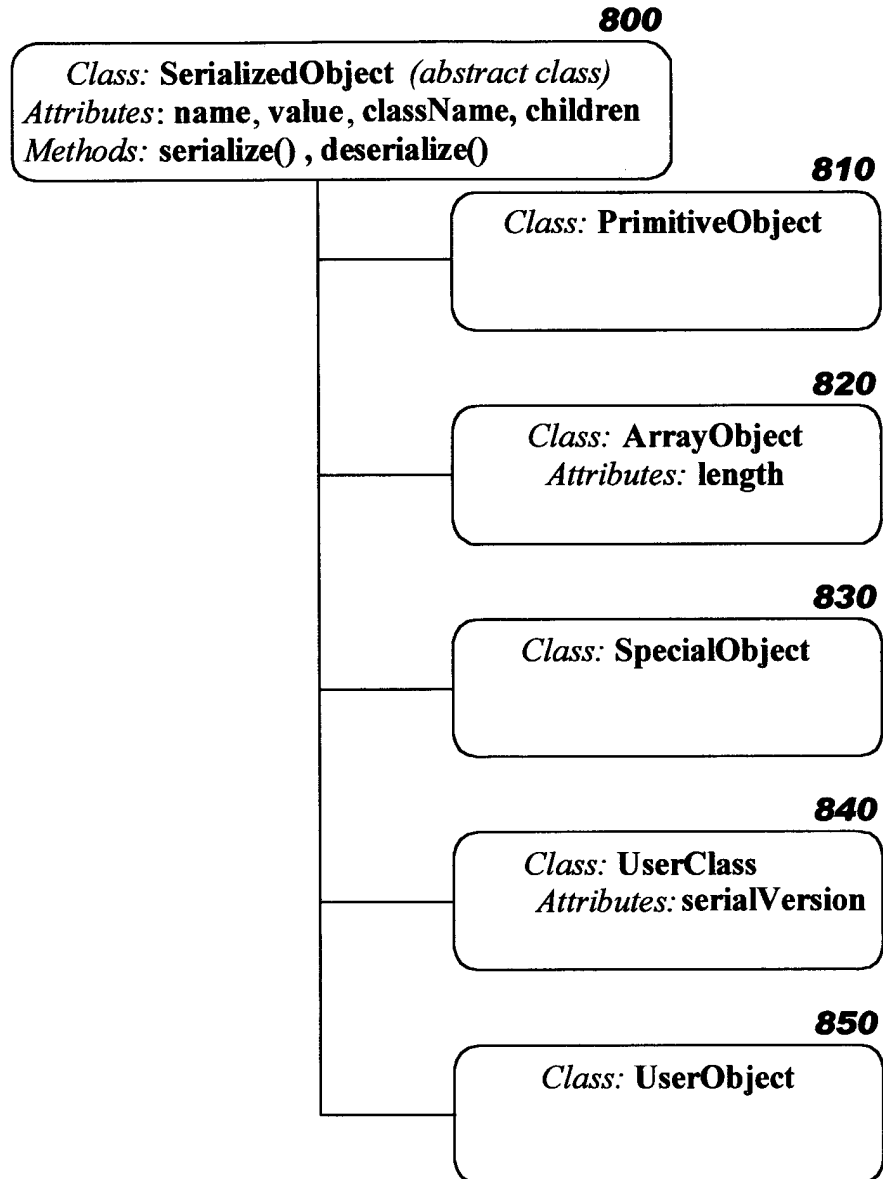
FIG. 7

**700**



6/27

FIG. 8



7/27

## FIG. 9

### **900**

#### **910** Class Person

```
{  
    public String name;  
    private int age;  
    ...  
}
```

...

#### **920** Class Employee extends Person

```
{  
    protected String title;  
    protected long officeNumber;  
  
    public Employee(String name, int age, String title, long officeNumber) { ... }  
    ...  
}
```

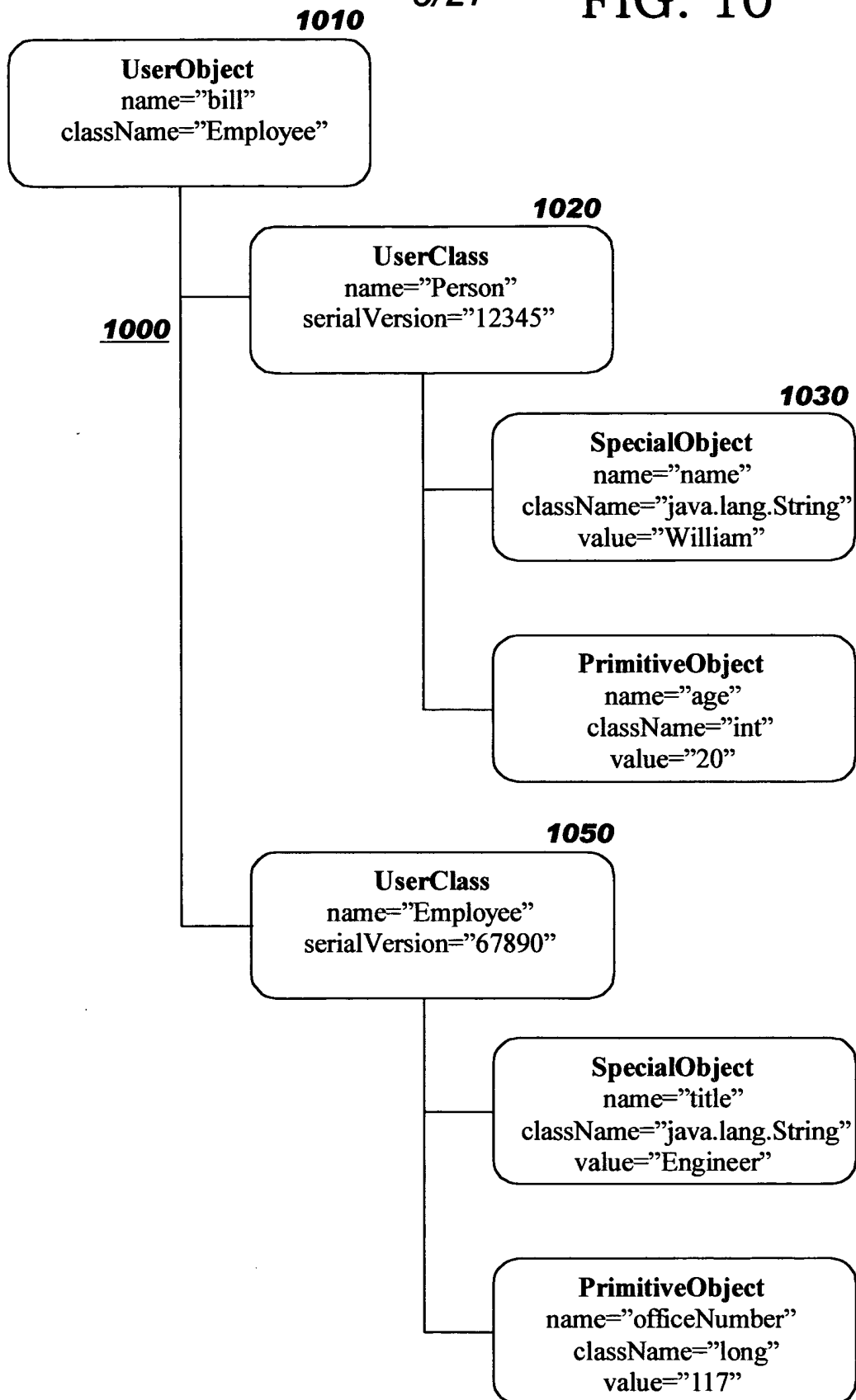
...

// Create a new instance of an employee

#### **930** Employee bill = new Employee("William", 20, "Engineer", 117);

8/27

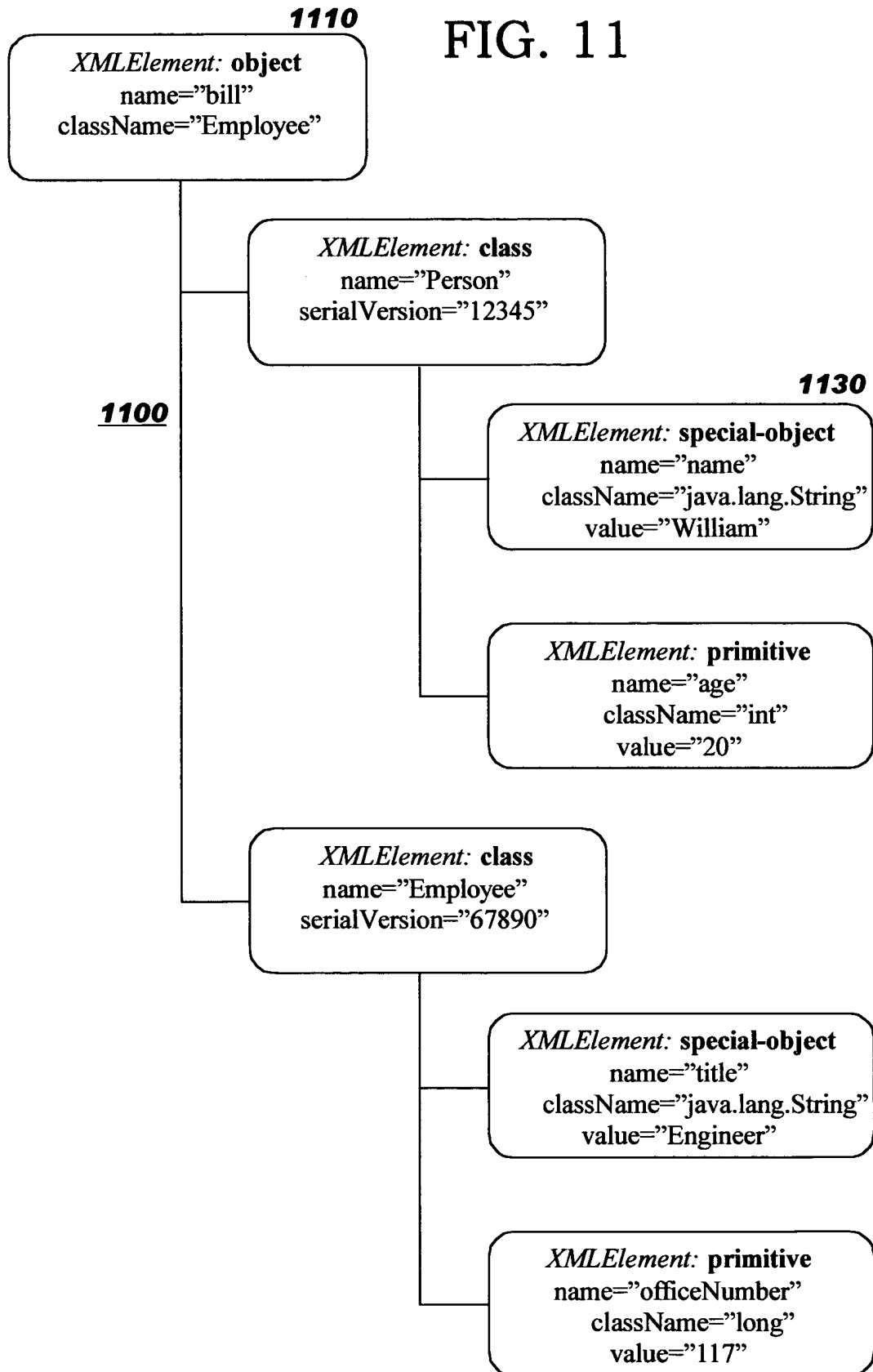
FIG. 10





9/27

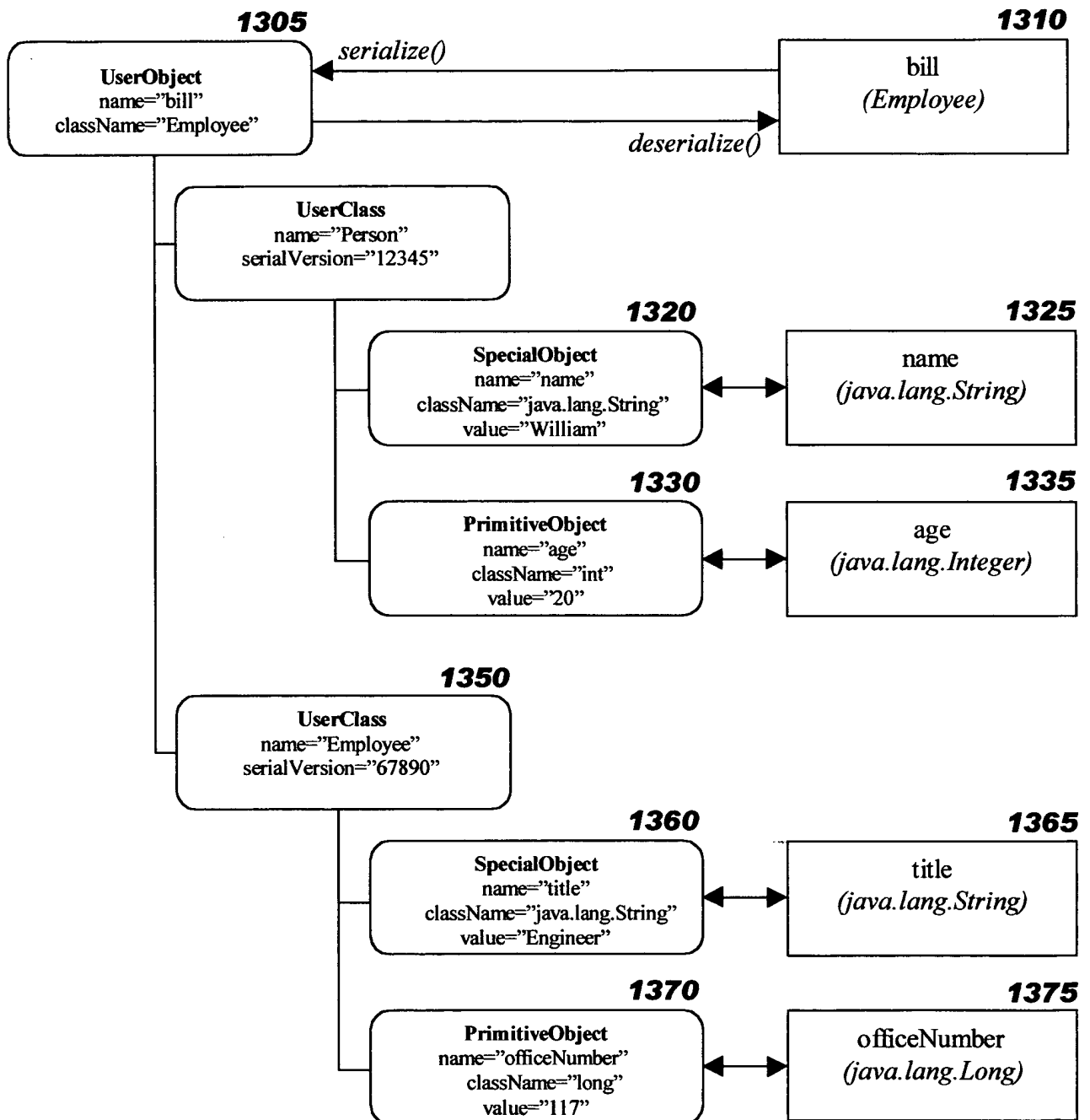
FIG. 11



10/27

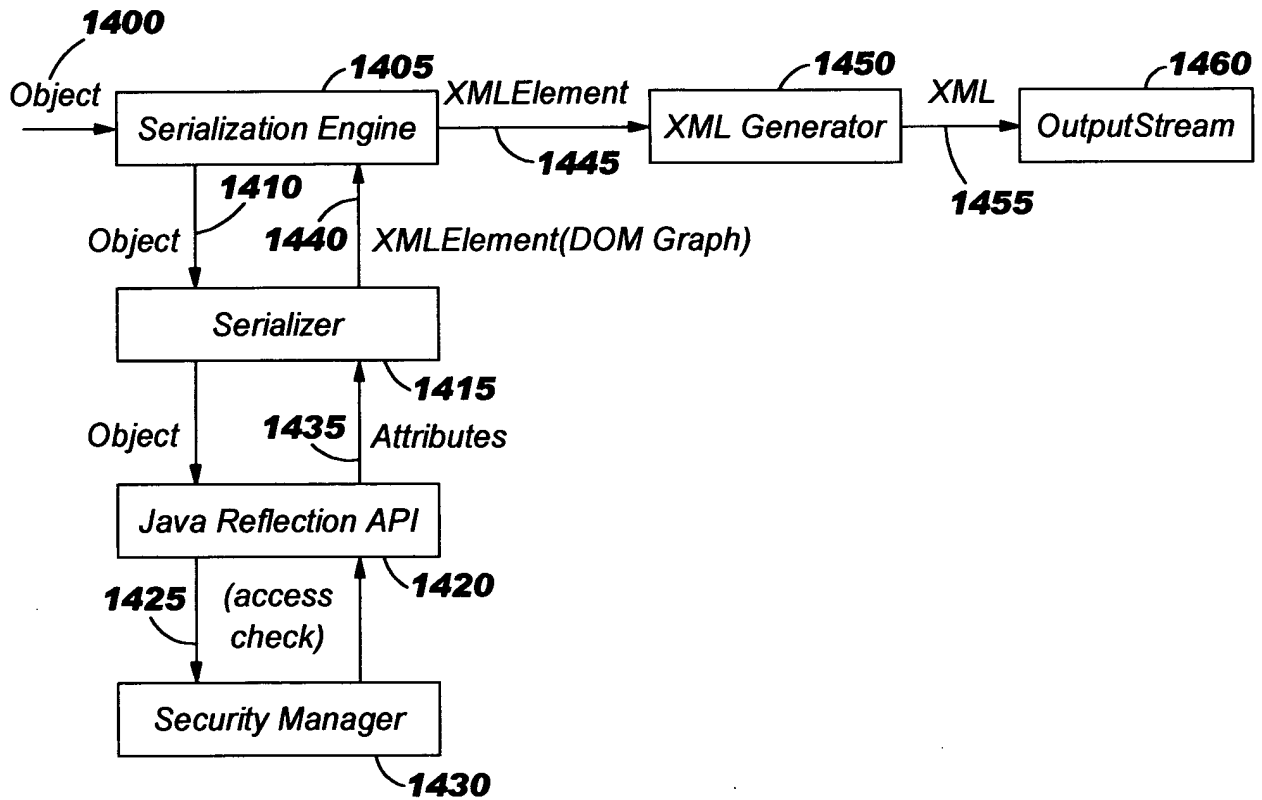
FIG. 13

**1300**



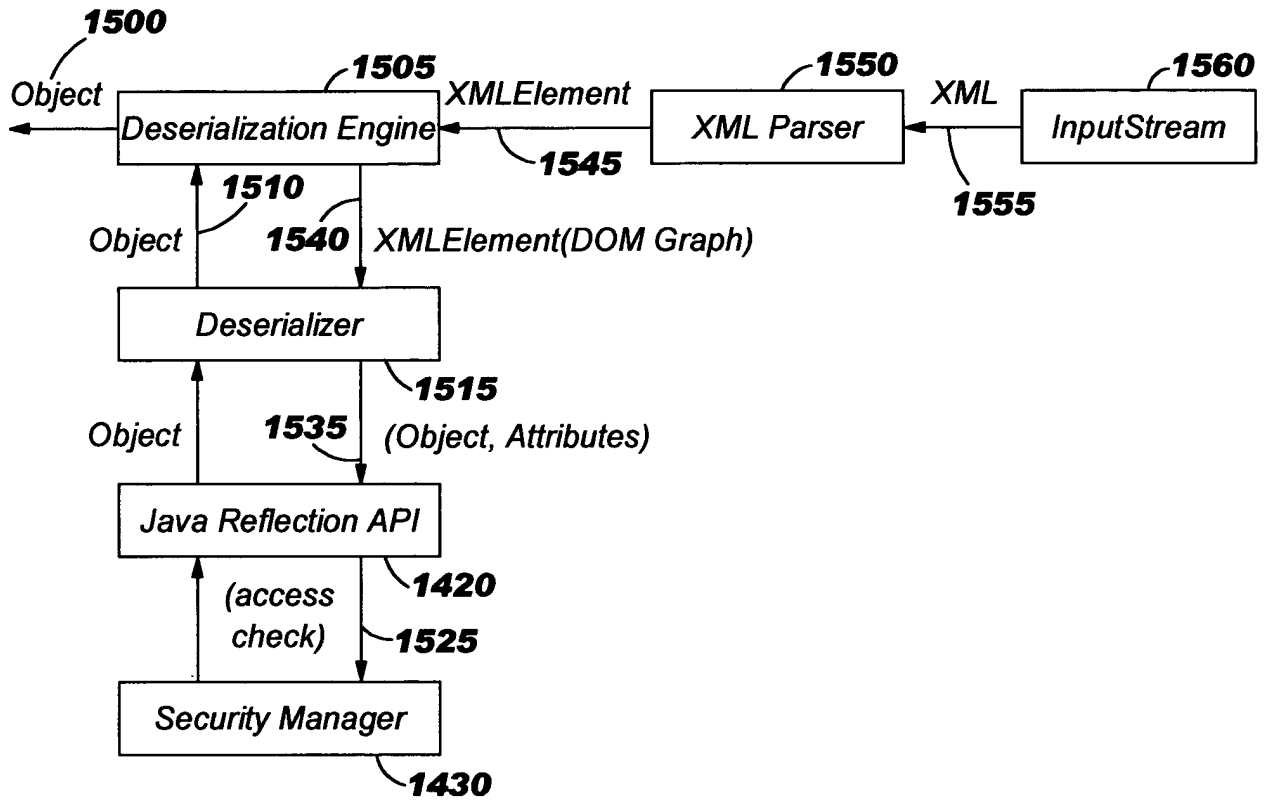
11/27

FIG. 14



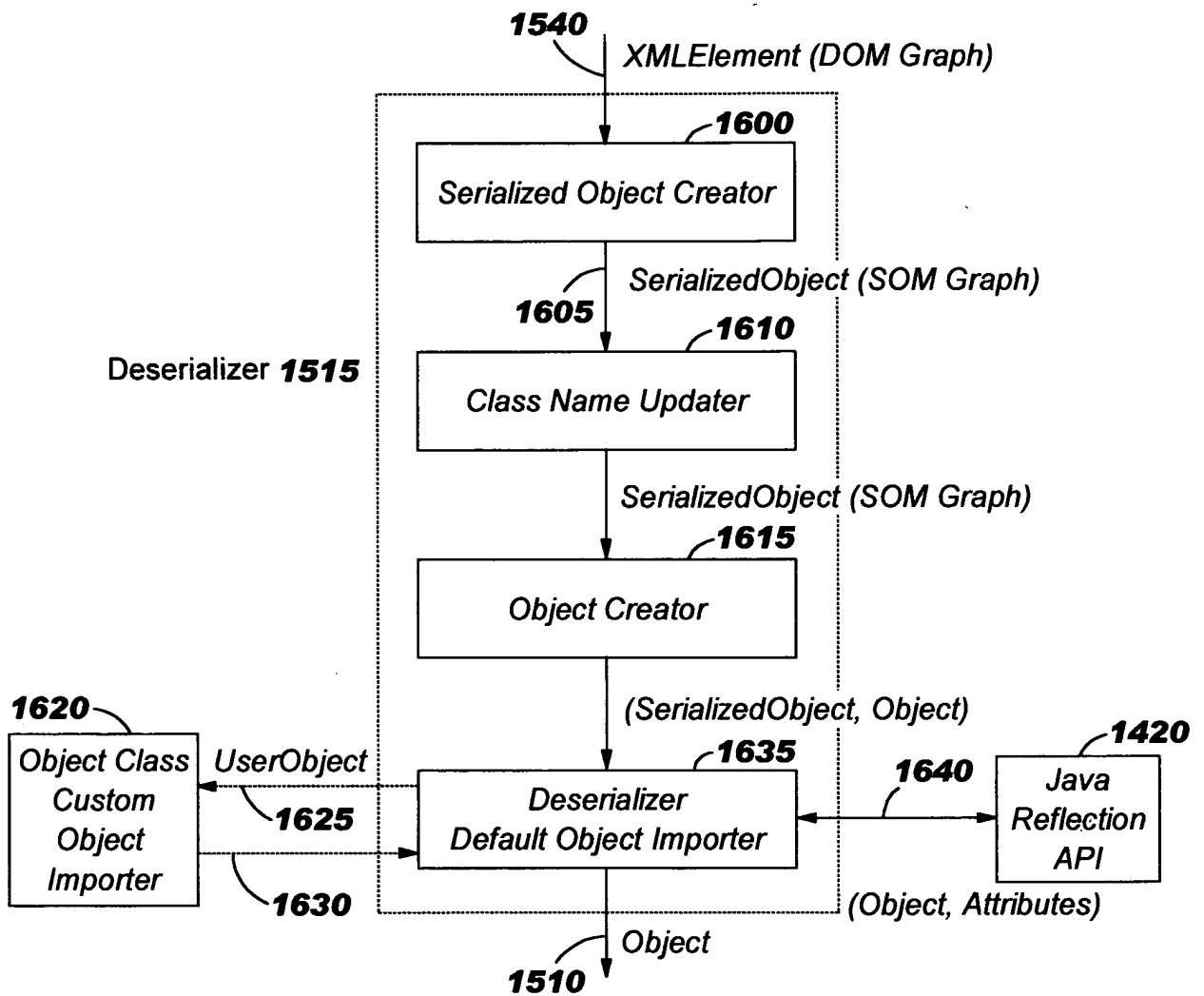
12/27

FIG. 15



13/27

FIG. 16



14/27

FIG. 17A  
(Prior Art)

**1700**

```
class ABC implements Serializable
```

```
{
```

```
int A;
```

```
boolean B;
```

```
...
```

```
}
```

FIG. 17B  
(Prior Art)

**1750**

```
class ABC implements Serializable
```

```
{
```

```
int A;
```

```
boolean B;
```

```
string C;
```

```
...
```

```
}
```

FIG. 17C  
(Prior Art)

**1790**

```
class ABC implements Serializable {
```

```
static final long serialVersionUID = 1111222233334444L;
```

```
int A;
```

```
boolean B;
```

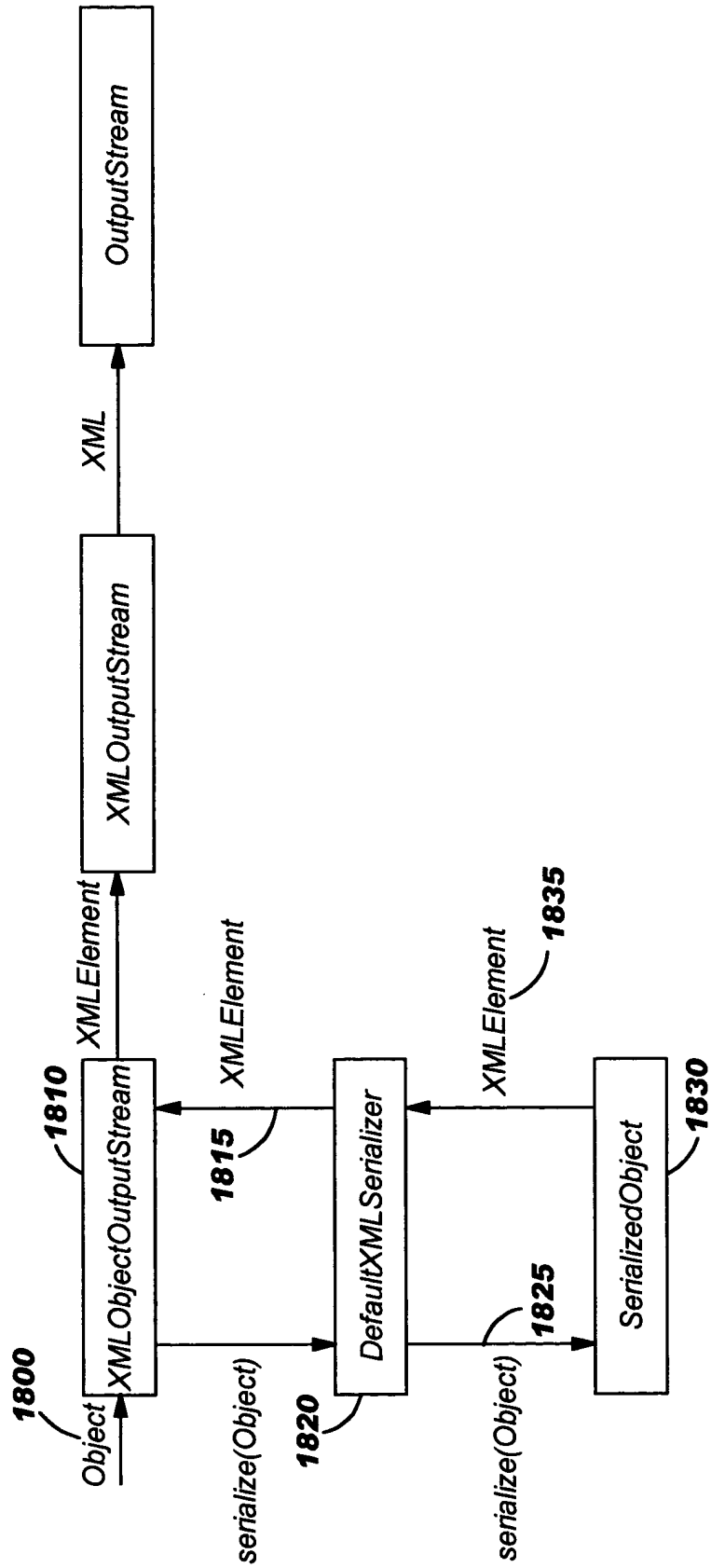
```
string C;
```

```
...
```

```
}
```

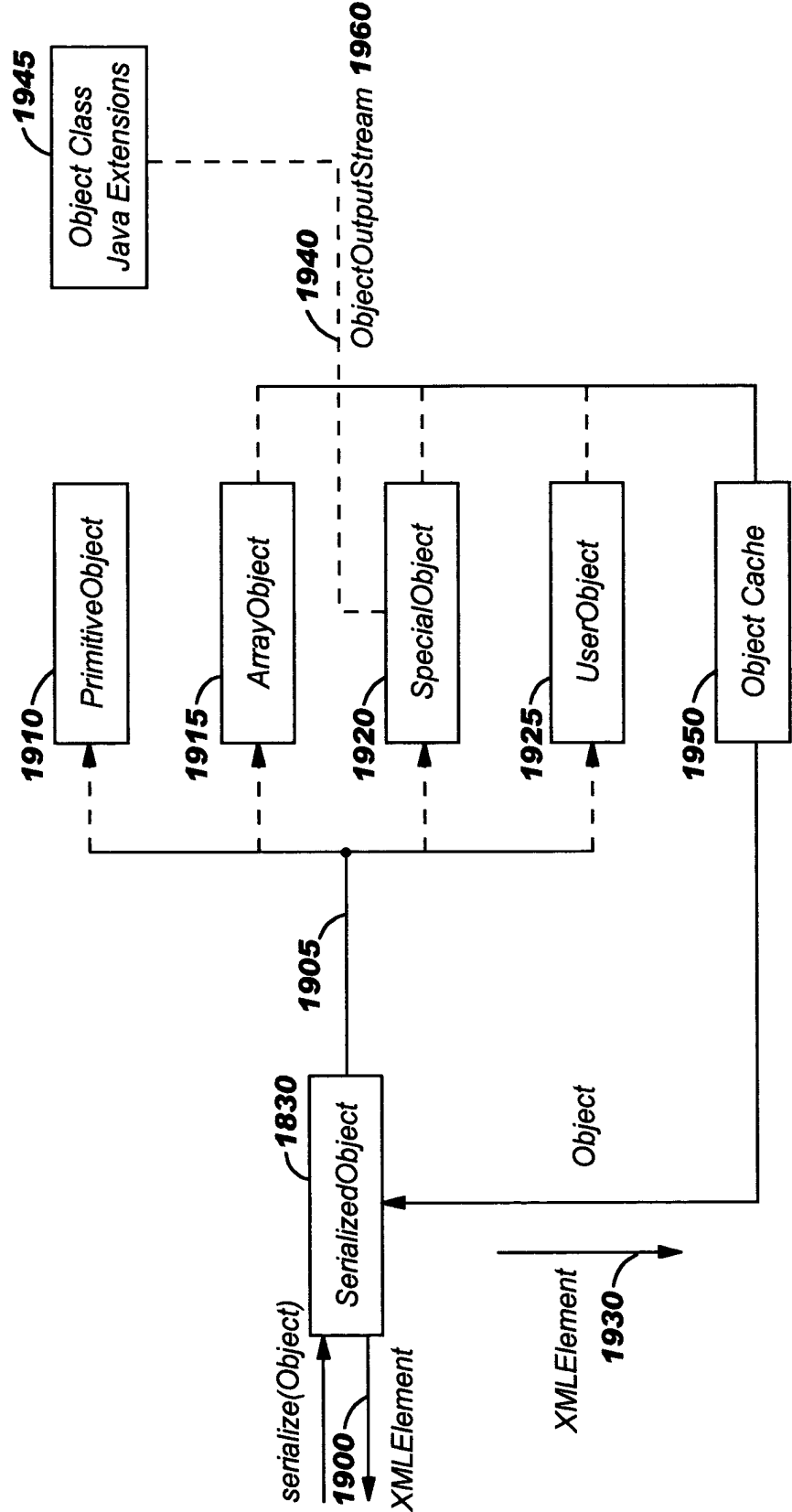
15/27

FIG. 18



16/27

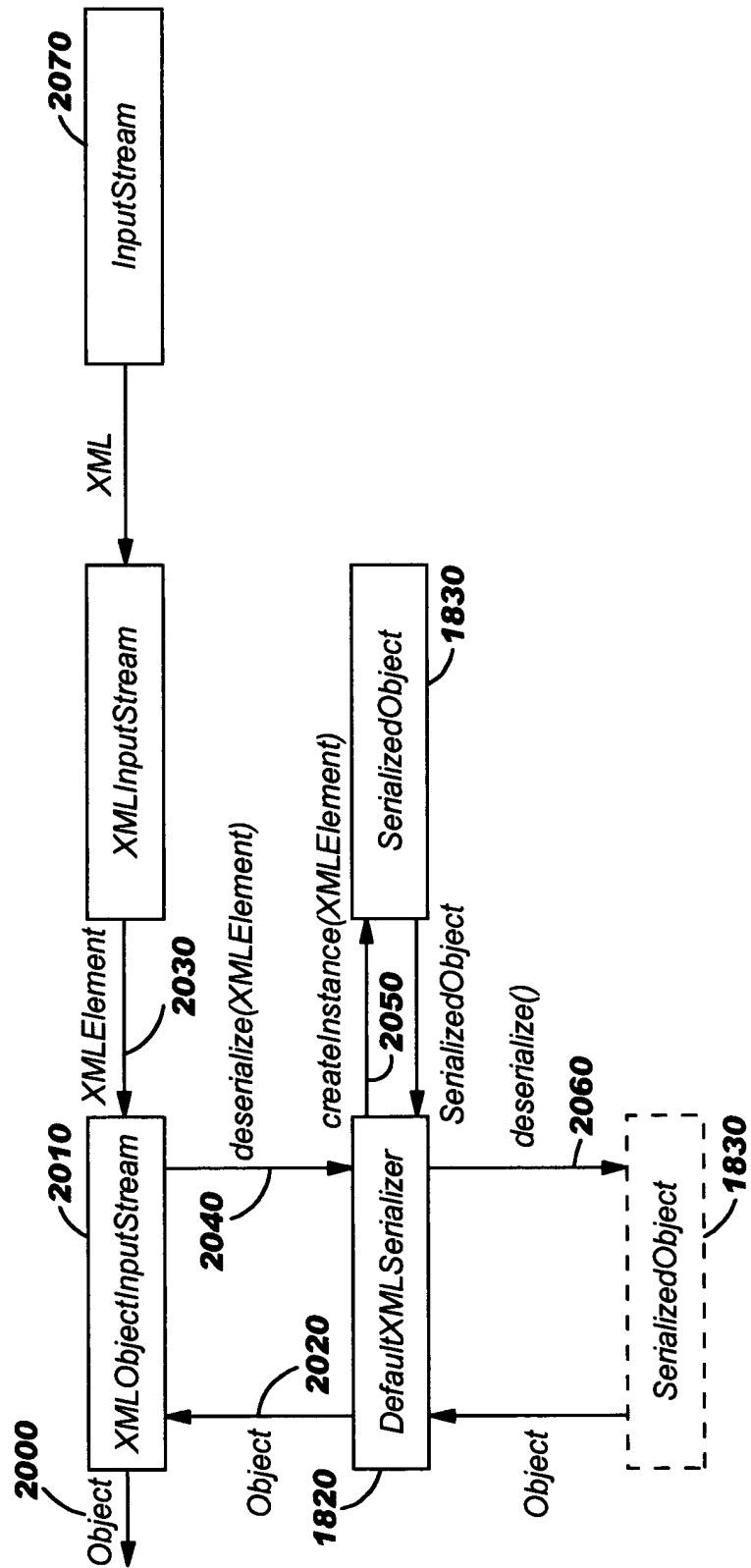
FIG. 19





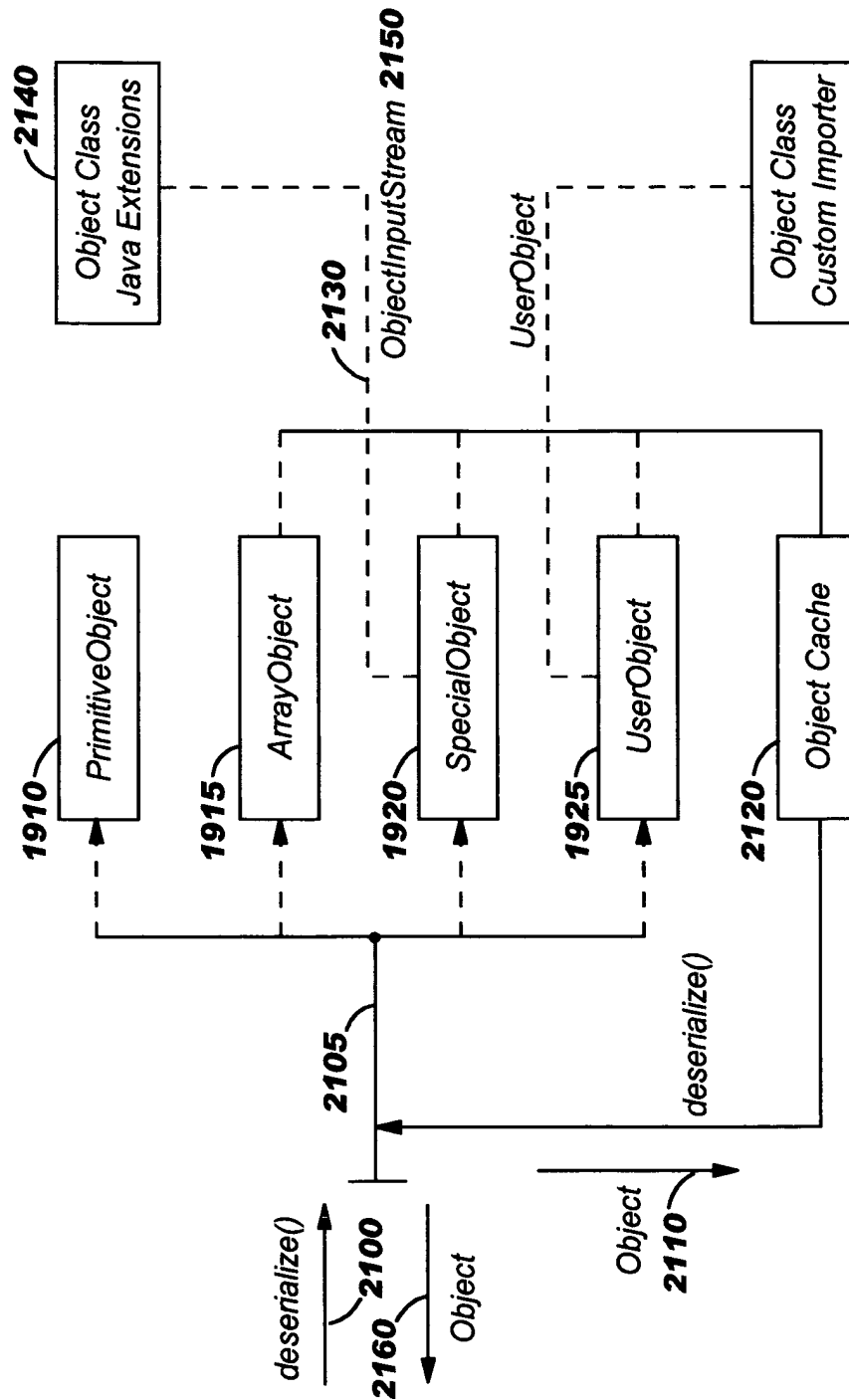
17/27

FIG. 20



18/27

FIG. 21



19/27

FIG. 22

```
2200 public abstract class Person
2210     implements Serializable, SerializedObjectImporter
    {
        protected String GUEST_ACCOUNT[ ] =
            new String[ ] { "unix", "guest", "secret" };

2220     private String name;
        private String address;
        private Vector accounts = new Vector();

        /** Constructor */
        public Person(String name, String address)
        {
            this.name = name;
            this.address = address;
            accounts.add(GUEST_ACCOUNT); 2230
        }

        /** Member functions */
        public String getName() { return name; }
        public String getAddress() { return address; }

        /**
         * Imports a serialized object into this object. This function is
         * called by the default deserializer for each object to customize
         * the import of the serialized object. The default object importer
         * may be called from this function to perform the default import.
         *
         * @param serialHandle Serialized object handle.
         * @param userObject   Serialized user object.
         */
2240     public void customObjectImport(Object serialHandle, UserObject userObject)
            throws XMLObjectStreamException, ClassNotFoundException
        {
            // Perform the default object import at the root class
            userObject.defaultObjectImport(serialHandle, this);

            // Get the serialized class entry for this class
            UserClass userClass = userObject.getClass(Person.class);

            // Perform manual import of changed and new fields
            if (userClass.isClassDifferent(this)) {}

        }
    } 2250
```

20/27  
FIG. 23

```
2300 public class Employee extends Person
{
    2310 private int officeNumber;
    private String emailAddress;
    private Date creationDate;
    private Vector roles = new Vector();

    /** Constructor */
2320 public Employee(String name, String address,
                    int officeNumber, String emailAddress)
    {
        super(name, address);
        this.officeNumber = officeNumber;
        this.emailAddress = emailAddress;
        this.creationDate = new Date();

        2330 // Create the roles
        this.roles.add("Administrator");
        this.roles.add("Manager");
    }

    /** Member functions */
    public int getOfficeNumber() { return officeNumber; }
    public String getEmailAddress() { return emailAddress; }
    public String getDateHired() { return creationDate.toString(); }

    /** Displays user information */
    public void displayUserInformation()
    {
        System.out.println("\nEmployee Information (Employee)");
        System.out.println("\tName: " + getName());
        System.out.println("\tAddress: " + getAddress());
        System.out.println("\tOffice number: " + getOfficeNumber());
        System.out.println("\tEmail address: " + getEmailAddress());
        System.out.println("\tDate Hired: " + getDateHired());
        System.out.println("\tRoles: " + roles.elementAt(0).toString() + ", "
            + roles.elementAt(1).toString());
    }
}
```

21/27  
FIG. 24

**2400**

```
import util.xml.stream.*;  
import util.xml.objectstream.*;
```

```
...
```

```
FileInputStream fsi;  
FileOutputStream fso;  
XMLObjectInputStream xsi;  
XMLObjectOutputStream xso;  
Employee employee;
```

```
// Create a new object
```

```
2410 employee = new Employee("Michael Smith", "2400 Oak Street", 17,  
                             "michael@internet.com");
```

```
2420 employee.displayUserInformation();
```

```
2430 [ // Write XML serialized object  
      fso = new FileOutputStream("output.xml");  
      xso = new XMLObjectOutputStream(fso);  
      xso.writeObject(employee);  
      xso.close();
```

```
2440 [ // Discard the object  
      employee = null;
```

```
2450 [ // Read XML serialized object  
      fsi = new FileInputStream("output.xml");  
      xsi = new XMLObjectInputStream(fsi);  
      employee = (Employee) xsi.readObject();  
      employee.displayUserInformation();  
      xsi.close();
```

22/27

## FIG. 25

### **2500**

#### **2510** Employee Information (Employee)

Name: Michael Smith  
Address: 2400 Oak Street  
Office number: 17  
Email address: michael@internet.com  
Date Hired: Thu Aug 14 14:30:30 EDT 2003  
Roles: Administrator, Manager

#### **2520** Employee Information (Employee)

Name: Michael Smith  
Address: 2400 Oak Street  
Office number: 17  
Email address: michael@internet.com  
Date Hired: Thu Aug 14 14:30:30 EDT 2003  
Roles: Administrator, Manager

23/27

**2600** **FIG. 26**  
<?xml version="1.0" encoding="UTF-8" ?>  
**2610** <serialized method="reflect-java-xml" version="1.0">  
**2620** <object class="Employee" reference="1">  
    **2631** <class name="Employee" version="-7349900520304056087">  
        <special-object name="roles" class="java.util.Vector" method="construct" reference="2">  
            <special-object class="java.lang.String" value="Administrator" method="construct" reference="3" />  
            <special-object class="java.lang.String" value="Manager" method="construct" reference="4" />  
        </special-object>  
**2630** <special-object name="creationDate" class="java.util.Date" method="construct" reference="5"  
        value="1060885830375" />  
    <special-object name="emailAddress" class="java.lang.String" value="michael@internet.com"  
        method="construct" reference="6" />  
    <primitive name="officeNumber" class="int" value="17" />  
    </class>  
    **2641** <class name="Person" version="-3933595787279955677">  
        <special-object name="accounts" class="java.util.Vector" method="construct" reference="7">  
            <array class="[Ljava.lang.String;" length="3" reference="8"> **2643**  
                <special-object class="java.lang.String" value="unix" method="construct" reference="9" />  
                <special-object class="java.lang.String" value="guest" method="construct" reference="10" />  
                <special-object class="java.lang.String" value="secret" method="construct" reference="11" />  
            </array>  
        </special-object>  
        <special-object name="address" class="java.lang.String" value="2400 Oak Street"  
            method="construct" reference="12" />  
        <special-object name="name" class="java.lang.String" value="Michael Smith"  
            method="construct" reference="13" />  
        <array name="GUEST\_ACCOUNT" class="[Ljava.lang.String;" length="3" reference="8" link="yes" />  
    </class>  
    </object>  
    </serialized>  
**2640** **2644**

24/27  
FIG. 27A

```
2700 public class NewEmployee extends Person 2710
{
    private String officeNumber;           // ** Modified type **
    //private String emailAddress;         // ** Deleted field **
2720 private String title;                   // ** Added field **
    private Date dateHired;                 // ** Renamed field **
    private String roles[ ];                // ** Modified type **

    /** Constructor */
2730 public NewEmployee(String name, String address, String officeNumber)
    {
        super(name, address);
        this.officeNumber = officeNumber;
        this.title = "Associate";
        this.dateHired = new Date();

        // Create the roles
        this.roles = new String[ ] { "Developer", "Tester" };
    }

    /** Member functions */
    public String getOfficeNumber() { return officeNumber; }
    public String getTitle()        { return title; }
    public String getDateHired()    { return dateHired.toString(); }

    /** Displays user information */
    public void displayUserInformation()
    {
        System.out.println("\nEmployee Information (NewEmployee)");
        System.out.println("\tName: " + getName());
        System.out.println("\tAddress: " + getAddress());
        System.out.println("\tOffice number: " + getOfficeNumber());
        System.out.println("\tTitle: " + getTitle());
        System.out.println("\tDate Hired: " + getDateHired());
        System.out.println("\tRoles: " + roles[0] + ", " + roles[1]);
    }
}
```



25/27  
FIG. 27B

```
/**
 * Imports a serialized object into this object. This function is
 * called by the default deserializer for each object to customize
 * the import of the serialized object. The default object importer
 * may be called from this function to perform the default import.
 *
 * @param serialHandle Serialized object handle.
 * @param userObject Serialized user object.
 */
public void customObjectImport(Object serialHandle, UserObject userObject)
    throws XMLObjectStreamException, ClassNotFoundException
{
    // Invoke super classes to perform class specific imports
    super.customObjectImport(serialHandle, userObject);

    // Get the serialized class entry for this class
    2750 userObject.changeClassName("Employee", "NewEmployee", true);
    UserClass userClass = userObject.getClass(NewEmployee.class);

    // Perform manual import of changed and new fields
    2760 if (userClass.isClassDifferent(this))
    {
        2770 officeNumber = userClass.getField("officeNumber").getValue();
        2780 title = "Engineer (" + userClass.getField("emailAddress").getValue() + ")";
        2790 dateHired =
            (Date) userClass.getField("creationDate").deserialize(serialHandle);

        // Convert the roles from a Vector to a String array
        Vector rolesData =
            (Vector) userClass.getField("roles").deserialize(serialHandle);
        2795 roles = (String[ ]) rolesData.toArray(new String[rolesData.size()]);
    }
}
```

26/27

## FIG. 28

### **2800**

```
// Open the XML file containing the previously serialized object
fsi = new FileInputStream("output.xml");

// Specifying the serializer allows partial or complete control
// over the serialization and deserialization of the object.
// The default serializer may be extended to update class names
// before deserialization. Specifying the default serializer
// is equivalent to not specifying it at all, with the present API.
xsi = new XMLObjectInputStream(fsi, new DefaultXMLSerializer());

// Read XML serialized object into a different class.
// If the default serializer was extended to change the class
// name from 'Employee' to 'NewEmployee', then the readObject()
// below could be called without any parameters.
2810 NewEmployee newEmployee =
    (NewEmployee) xsi.readObject(new NewEmployee("", "", ""));
newEmployee.displayUserInformation();
```

## FIG. 29

### **2900**

Employee Information (NewEmployee)

Name: Michael Smith

Address: 2400 Oak Street

Office number: 17

**2910** Title: Engineer (michael@internet.com)

Date Hired: Thu Aug 14 14:30:30 EDT 2003

Roles: Administrator, Manager

## 27/27 FIG. 30

### Grammar

stream ::= header serialized-object

serialized-object ::= element("serialized", object)

object ::= user-object | special-object | array | primitive

user-object ::= element("object", 0\*user-class)

user-class ::= element("class", 0\*object)

special-object ::= element("special-object", 0\*object)

array ::= element("array", 0\*object)

primitive ::= element("primitive")

### Literals

header ::= <XML version control header>

element(tagname) ::= <XML element with specified  
tag name and zero children>

element(tagname,child) ::= <XML element with specified  
tag name and child elements>

### Tag Attributes

serialized: method version

class: name version

object: [name] class reference [link]

array: [name] class reference [link] length

special-object: [name] class [value] reference [link] method

primitive: [name] class value